APACHE GERONIMO

J2EE Development with Apache Geronimo

Aaron Mulder
Chariot Solutions
ammulder@chariotsolutions.com

Speaker

- Aaron Mulder
- Geronimo Developer
  - Works on deployment, management, console, kernel, ...
  - Online Geronimo book at `http://chariotsolutions.com/geronimo/`
- CTO of Chariot Solutions
  - Java/Open Source consulting firm
  - Partnerships with companies that provide Geronimo support (IBM, Covalent, etc.)

Agenda

- Server Installation and Configuration
- Deployment Tools
- Configuring J2EE Applications
- Q&A

Installation and Configuration

Installation

- .ZIP & .TAR.GZ distributions available now
- ZIP/TAR: Download and unzip either the Jetty or the Tomcat distribution
- Edit ports in `var/config/config.xml` (more on this in a bit)
  - Have to get it running before the console is available
- Installer: run `java -jar geronimo-installer.jar` and make your selections accordingly

Start and Stop

- Start: run `java -jar bin/server.jar`
  - command-line options:
    - --long (simpler startup output)
    - --quiet (no progress bar)
    - -v or -vv (more log output to console)
- Stop: `Ctrl-C` or `java -jar bin/shutdown.jar`

Startup Sequence

```
Booting Geronimo Kernel (in Java 1.4.2_09)...
Starting Geronimo Application Server
[**************] 100% 18s Startup complete

  Listening on Ports:
    1099 0.0.0.0 RMI Naming
    1527 0.0.0.0 Derby Connector
    4201 0.0.0.0 ActiveIO Connector EJB
    4242 0.0.0.0 Remote Login Listener
    8080 0.0.0.0 Jetty Connector HTTP
    8443 0.0.0.0 Jetty Connector HTTPS
    61616 0.0.0.0 ActiveMQ Message Broker Connector
  Started Application Modules:
    EAR: org/apache/geronimo/Console/Jetty
    WAR: org/apache/geronimo/applications/Welcome/Jetty
  Web Applications:
    http://server-hostname:8080/
    http://server-hostname:8080/console
    http://server-hostname:8080/console-standard

Geronimo Application Server started
```

Configuration (easy)

- Start server and point browser to `http://localhost:8080/console/`
- Use the screens there to edit network ports, add database connection pools, etc.
- May need to restart the server to apply certain changes
- Can't use if original network ports conflict
  - Use the next option to resolve ports and then go into the console. :)

Configuration (hard)

- Most configuration is controlled by `config.xml` in `var/config`
  - Controls which configurations to load
  - Lets you override settings on any server component (identified by config name + component name + attribute name)
- Note that the server rewrites this file while it's running
  - Edit it only while the server is down!

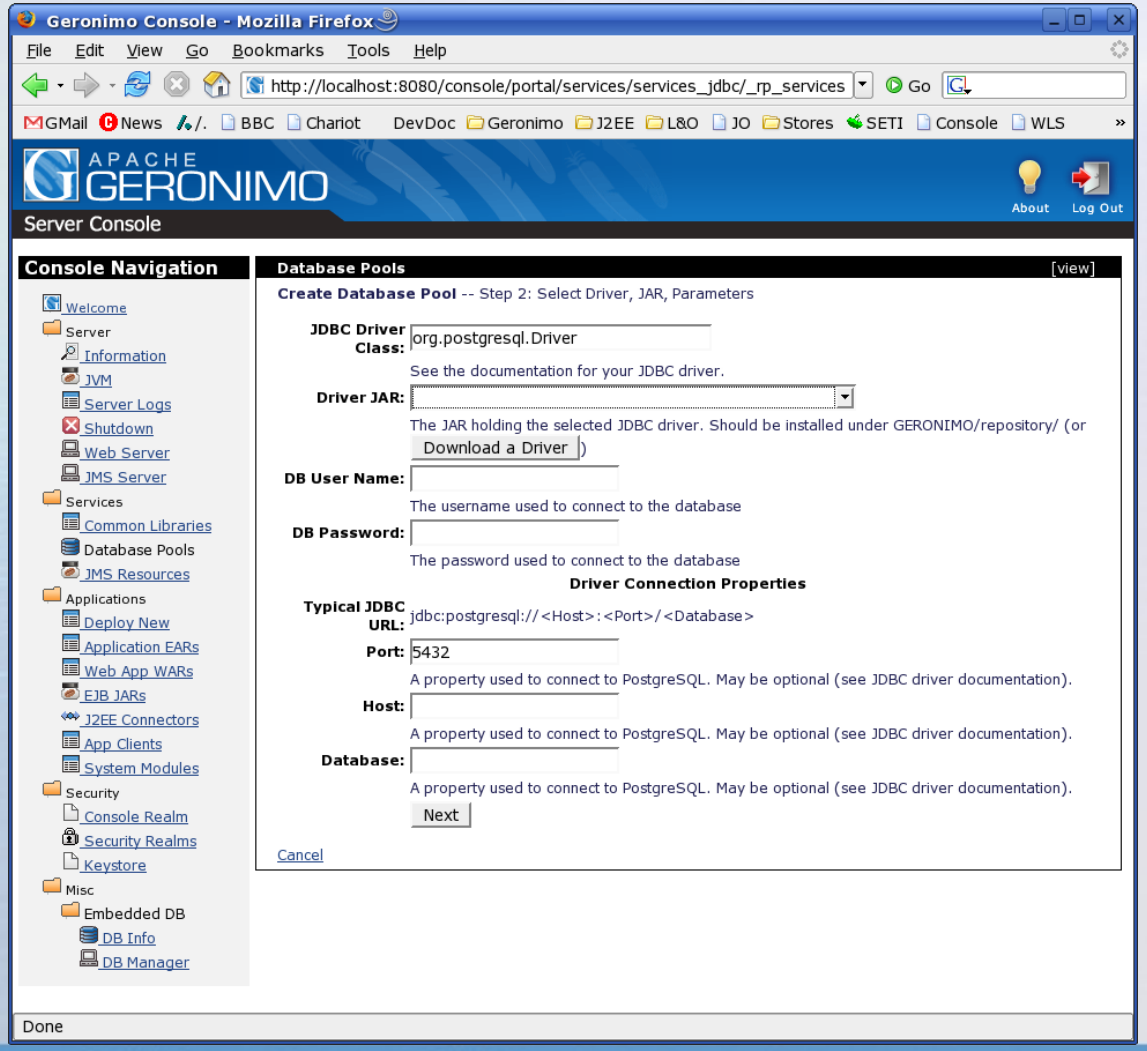
config.xml

```
<attributes
xmlns="http://geronimo.apache.org/xml/ns/attributes">
  <configuration name="geronimo/rmi-naming/1.0/car">
    <gbean name="RMIRegistry">
      <attribute name="port">1099</attribute>
    </gbean>
    <gbean name="NamingProperties">
      <attribute name="namingProviderUrl">
        rmi://0.0.0.0:1099
      </attribute>
    </gbean>
  </configuration>
  <configuration name= ...  />
  ...
</attributes>
```

Logging

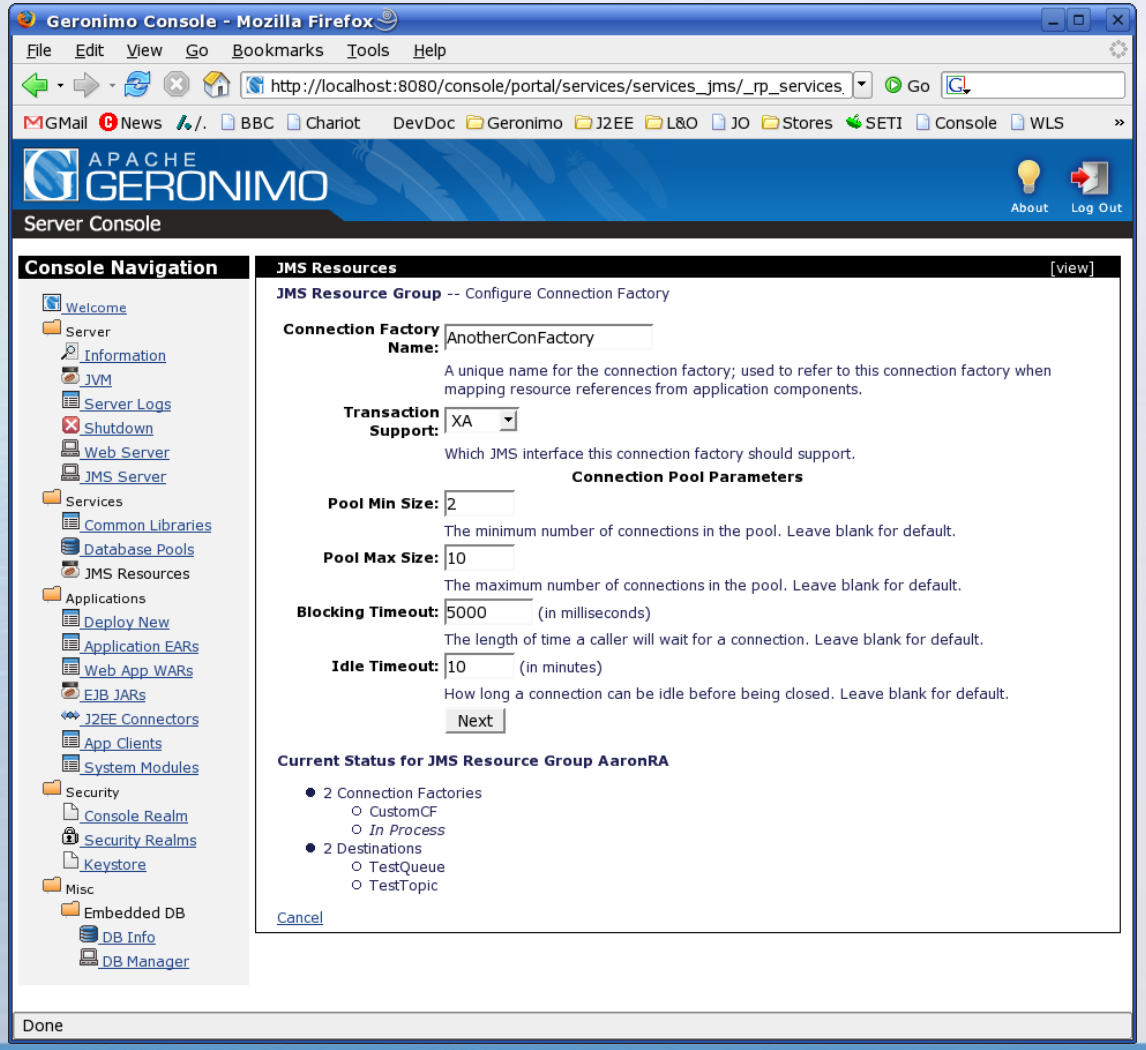
- Uses Log4J
- Config file at `var/log/server-log4j.properties`
- Server log at `var/log/geronimo.log`
- Console log level defaults to INFO (reduce with -v or -vv on startup)
- Can search server log and web access logs in the console (though not as fast as grep)

Database Pools



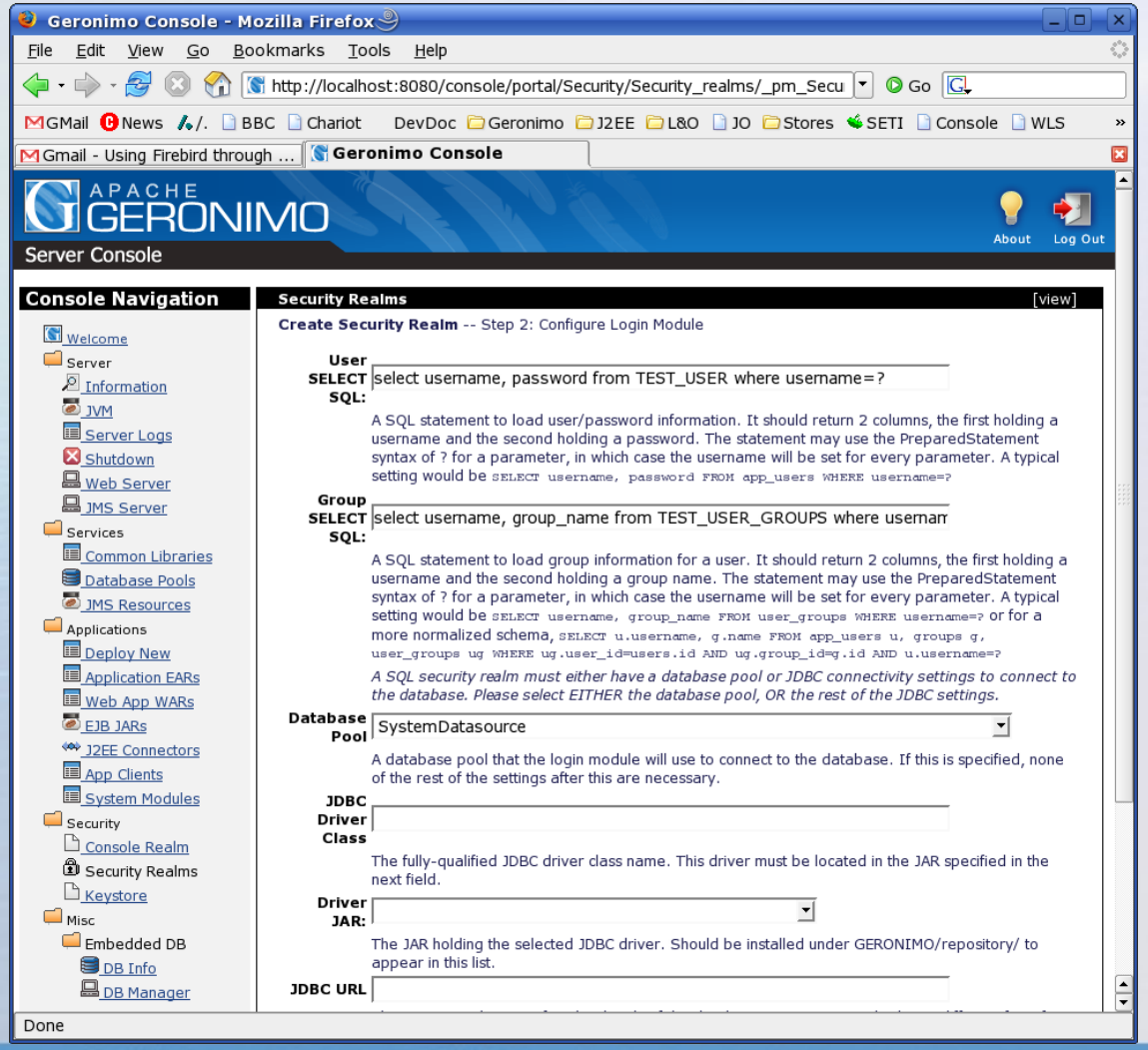
- Can deploy by hand
- Can deploy as part of an application
- Options include pool size, exception handler, etc.

JMS Resources



- Geronimo starts an ActiveMQ broker by default
- Can also deploy by hand or as part of an application

Security Realms

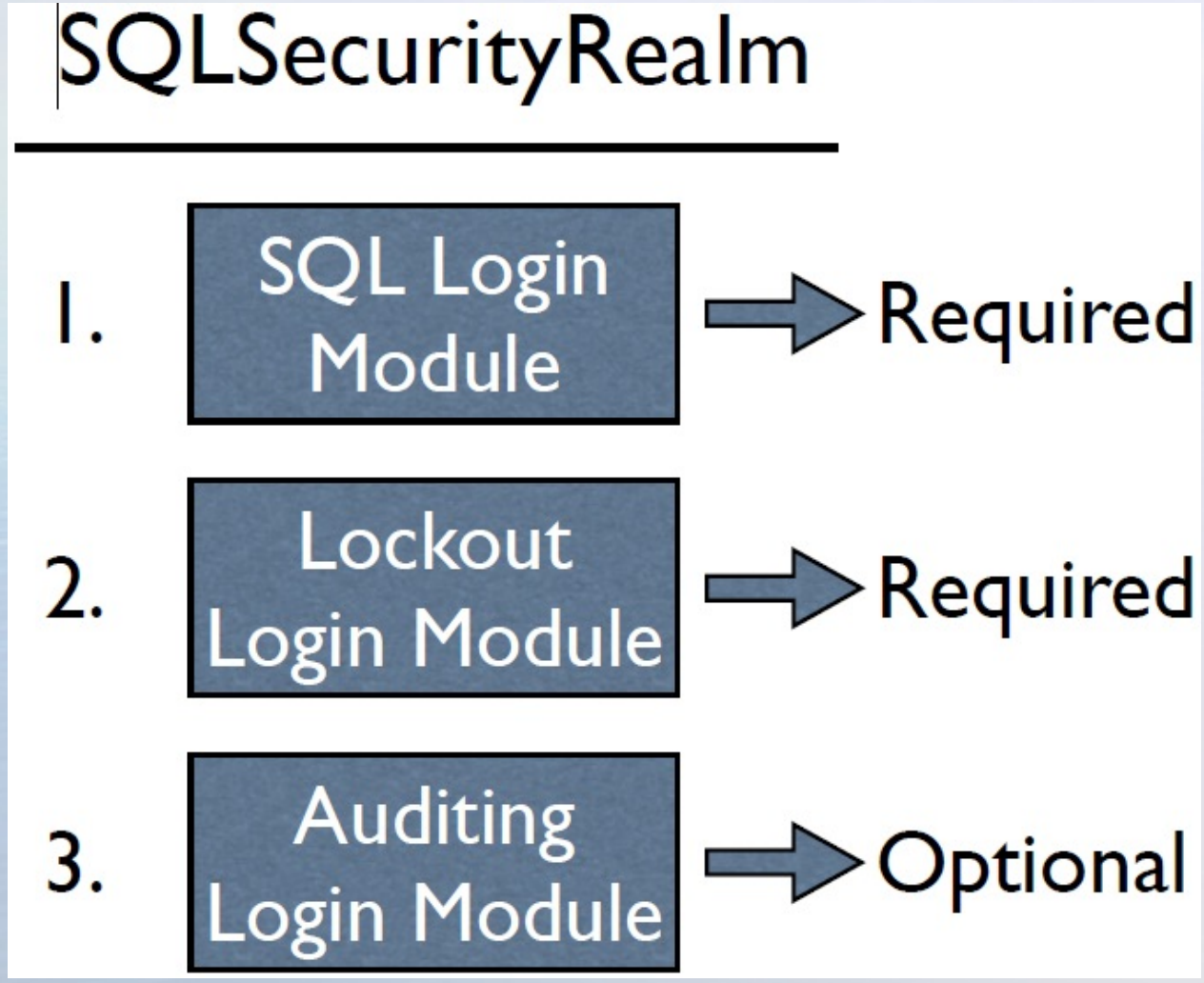


- Based on JAAS LoginModules
- Can deploy by hand or as part of an application
- Default in `var/security` properties

JAAS Login Modules

- A realm normally uses one LoginModule, but may include several
- Extra features are added by using multiple LoginModules for the realm
  - auditing, lockout, extra credentials, etc.
- When mapping security later, you'll need to know what classes the LoginModules use to represent the Principals (users/groups)

Realm Example



Included Login Modules

- Properties File
- Kerberos
- LDAP
- SQL
- Auditing
- Lockout on repeated failure
- Save credentials to use when invoking a web service or CORBA EJB

Deployment

Deployment Overview

- For apps: need an archive or directory with a J2EE deployment descriptor, and typically a Geronimo deployment plan
- For services (custom configurations): just need a Geronimo deployment plan
- Use the deploy tool, maven plugin, or hot deploy directory to deploy the app or service
  - Deploy tool and Maven plugin return errors and a success code to the caller; better for scripting

Deployment Plan

- aka “server-specific deployment descriptor”
- Geronimo plans are based on XML Schemas (normally one per module type)
- Schemas can be found in schemas/
- Always have a `configId` (a unique ID for the module) and optional `parentId` and `include`’s (used to set up class loaders and startup dependencies)

Typical Deployment Plan

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
 xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.0"
 configId="geronimo/jmxdebug/1.0/car"
 parentId="geronimo/j2ee-server/1.0/car">

   <dependency>
     <uri>
       commons-collections/commons-collections/3.1/jar
     </uri>
   </dependency>

   <context-root>/debug-tool</context-root>
   <context-priority-classloader>
     false
   </context-priority-classloader>
</web-app>
```

Digression: Namespaces

- Several part of the plan (typically the ones reused across many plan types) come from different namespaces
- You can write your files all in the owning plan's namespace, and Geronimo will be fine with that (but XML editors may not)
- You can use the correct namespaces and your XML editor will be happier and Geronimo will be fine with that too

Strictly Correct Plan

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
 xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.0"
 xmlns:dep=
    "http://geronimo.apache.org/xml/ns/deployment-1.0"
 configId="geronimo/jmxdebug/1.0/car"
 parentId="geronimo/j2ee-server/1.0/car">

   <dep:dependency>
     <dep:uri>
       commons-collections/commons-collections/3.1/jar
     </dep:uri>
   </dep:dependency>

   <context-root>/debug-tool</context-root>
   ...
```

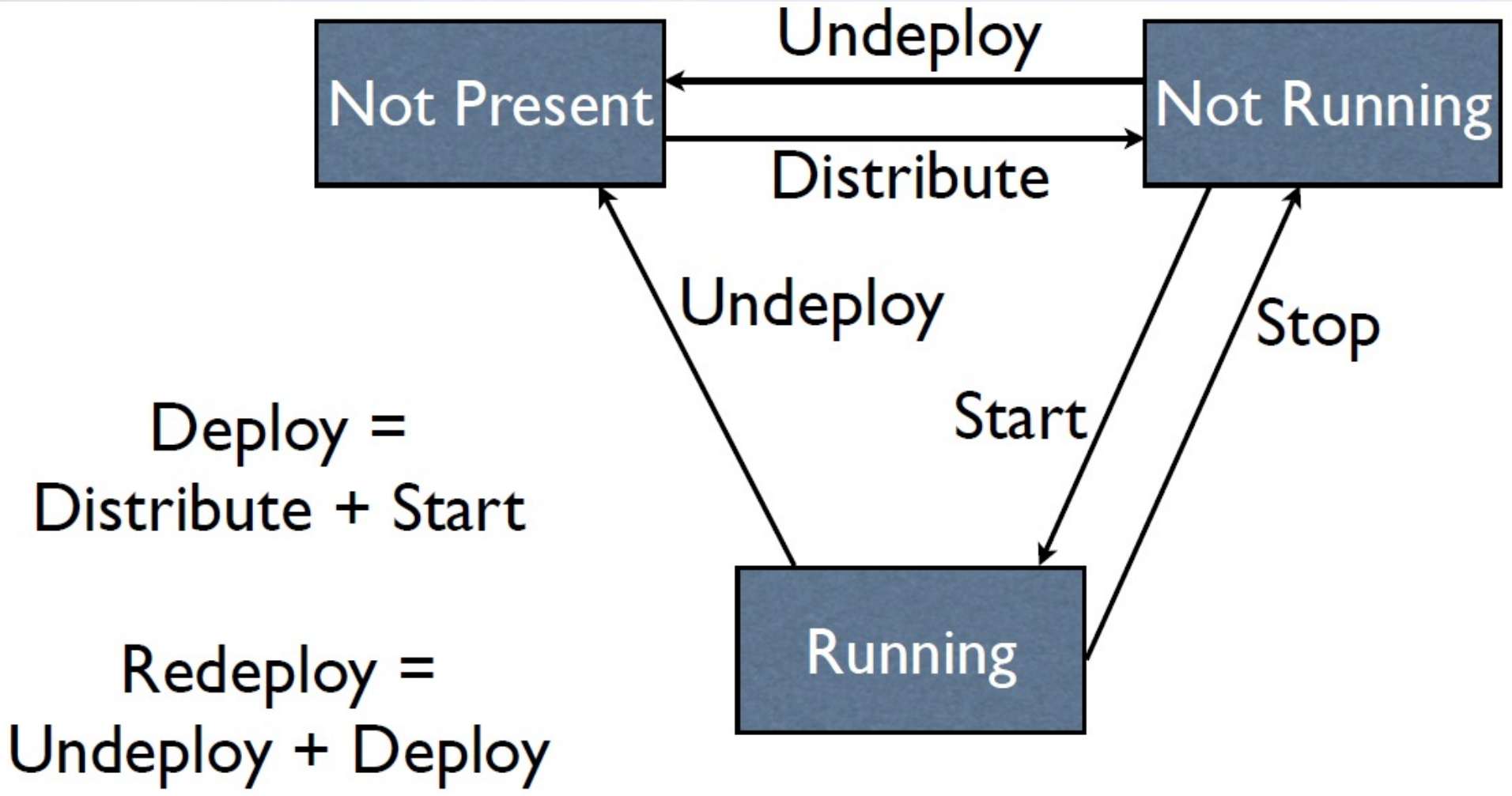
Command-Line Deploy Tool

- Communicates with a running server
- Run with `java -jar bin/deployer.jar [options] command [command-options]`
- Commands include `login`, `help`, `deploy`, `undeploy`, `redeploy`, `list-modules`, etc.
- Normally prompted for a username and password ("system" and "manager" unless you selected something different)
  - see `var/security/*.properties`

Remote Deployment

- Deploy tool can manage and deploy to a remote server
- Need to be able to access the RMI port (1099) and an HTTP(S) port (8080)
- Must have the remote-deploy web application deployed on the server
  - It is deployed by default
- use `--host` and `--port` (or perhaps `--uri`)

Module Lifecycle



Deploy =
Distribute + Start

Redeploy =
Undeploy + Deploy

Sample Commands

- java -jar bin/deployer.jar ...
  - login
  - distribute [archive] [plan]
  - deploy [archive] [plan]
  - undeploy configId
  - redeploy [archive] [plan] [configId]
  - stop configId
  - start configId
  - list-modules

Config IDs

- When you deploy, you'll get output like:

```
Deployed geronimo/webconsole-jetty/1.0/car
```

- That is the Config ID for the module, used to start, stop, undeploy, or redeploy it
- It is set by the `configId` attribute in the deployment plan
  - Defaults to the JAR name if no plan is provided

In context...

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
 xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.0"
 configId="geronimo/webconsole-jetty/1.0/car"
 parentId="geronimo/j2ee-server/1.0/car">
...
```

```
> java -jar bin/deployer.jar deploy console.war
Deployed geronimo/webconsole-jetty/1.0/car
```

```
> java -jar bin/deployer.jar stop
                  geronimo/webconsole-jetty/1.0/car
```

```
> java -jar bin/deployer.jar list-modules
Found 33 modules
  geronimo/webconsole-jetty/1.0/car
  ...
```

Parent IDs

- The optional `parentId` attribute controls the ClassLoader structure and startup order
- Can additionally specify `import` elements in the body of the deployment plan
- For startup order, can also just deploy a DB pool or JMS resource as part of an EAR
- Typical value is `geronimo/j2ee-server/1.0/car`
- Do not explicitly set this unless you need to
  - Can lock you to a specific version of the parent

Hot Deploy Directory

- `geronimo/deploy/`
- Copy files to this directory to deploy
  - update file to redeploy
  - delete file to undeploy
- On startup, recognizes new deployments, but will not undeploy or redeploy
- Should use either command-line deployer or hot deployer for any given module
  - That is, don't mix techniques for one module

Maven Plugin

- Deployment plugin for Maven 1.x can start & stop server, deploy/undeploy/redeploy applications, start server and wait until it runs, etc.
- Can be included in build scripts and won't return until application is running (for subsequent testing, etc.)
- Maven 2 & Ant plugins should be coming in 1.2 or 2.0s

Eclipse Plugin

- Works with Eclipse WTP
- Can create Geronimo apps, including XDoclet-based EJBs, etc.
- Can run an embedded Geronimo server
- Can deploy to Geronimo
- Can debug the embedded Geronimo
- Not quite in a “finished” state, but working

Debugging

- In IDEA, create a new debugging configuration and select “Remote”
- IDEA gives you a bunch of command-line parameters; start Geronimo with those

```
java -Xdebug -Xnoagent... -jar bin/server.jar
```

- Then remote connection works perfectly
- Eclipse can run and debug Geronimo locally
- Should be able to debug both the server (if you have the source) and applications

Common Deployment Plan Elements

Plans, revisited

- Generally hold things like:
  - Classloader/dependency configuration
  - Security mapping
  - Database/JMS/EJB/Web Service reference mapping
  - Component-specific configuration
    - EJB CMP, RA config settings, etc.
  - Custom services (GBeans)
- Required if any of the mapping is necessary

Common elements

- <dependency> lists a JAR that should be added to the module's class loader
- The JAR must be in `geronimo/repository`
- The "uri" is in the repository format of *groupId/artifactId/version/type*, like the standard `geronimo/j2ee-server/1.0/car`
- <gbean> lists custom services to be loaded when this module is loaded

More Common elements

- <security> holds security mapping (which users/groups are in which J2EE roles)
- <ejb-ref>, <ejb-local-ref>, <resource-ref>, <resource-env-ref> hold more mapping
  - Even can do CORBA, in the case of remote EJBs
- Doesn't use JNDI, uses a combination of the app name and component name
- <service-ref> resolves Web Services clients

3rd Party JAR Example

File at geronimo/repository/postgresql/jars/
postgresql-8.0-313.jdbc3.jar

```
<dependency>
  <uri>postgresql/postgresql-8.0/313.jdbc3/jar</uri>
</dependency>
```

```
<dependency>
  <groupId>postgresql</groupId>
  <type>jar</type>
  <artifactId>postgresql-8.0</artifactId>
  <version>313.jdbc3</version>
<dependency>
```

Component Mapping

- Need a name to identify the reference we're resolving, then one of a:
  - link (short name identifying the target, in same application or top-level in server)
  - "target-name" (long name uniquely identifying the target anywhere in server)
  - group of elements containing all the components of the target-name
    - Typically used to refer to components in another module or application

Component Example

```
<resource-ref>
  <ref-name>jdbc/MyDatabase</ref-name>
  <resource-link>PGSQLPool</resource-link>
</resource-ref>
```

```
<resource-ref>
  <ref-name>jdbc/MyDatabase</ref-name>
  <target-name>geronimo.server:J2EEApplication=null,
J2EEServer=geronimo,JCAResource=PostgreSQLPoolConfigID,
j2eeType=JCAManagedConnectionFactory,name=PGSQLPool
  </target-name>
</resource-ref>
```

```
<resource-ref>
  <ref-name>jdbc/MyDatabase</ref-name>
  <module>PostgreSQLPoolConfigID</module>
  <type>JCAManagedConnectionFactory</type>
  <name>PGSQLPool</name>
</resource-ref>
```

Security Mapping

- Security settings declared at the application level (EAR) apply to all included modules
- Map principals (by principal class and name) to J2EE Roles
- Indicate a default principal to use when the user does not authenticate
- Indicate a principal to use whenever a runas role applies

Security Mapping Example

```
<security>
  <default-principal>
    <principal name="nobody"
class="org.apache.geronimo.security.realm.providers.Ger
onimoUserPrincipal" />
  </default-principal>
  <role-mappings>
    <role role-name="Administrators">
      <principal name="Admins"
class="org.apache.geronimo.security.realm.providers.Ger
onimoGroupPrincipal" />
      <principal name="Aaron"
class="org.apache.geronimo.security.realm.providers.Ger
onimoUserPrincipal" />
    </role>
  </role-mappings>
</security>
```

Sample J2EE Module Plans

Web Applications

- Plan in WAR at `WEB-INF/geronimo-web.xml`
- Web settings for context path, classloader configuration (parent-first vs. WAR-first), security realm used to validate logins
- Container-specific virtual host settings
- Otherwise pretty standard (dependencies, resource/EJB/service references, security...)

WEB-INF/geronimo-web.xml

```
<web-app
 xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.0"
 configId="MyWebAppName">

  <dependency ...>
  <context-root>/debug-tool</context-root>
  <context-priority-classloader>
    false
  </context-priority-classloader>
  <container-config ...>
  <ejb-ref ...> <service-ref ...> <resource-ref ...>
  <security-realm-name>SQLRealm</security-realm-name>
  <security ...>
  <gbean ...>
</web-app>
```

EJB JARs

- Plan in JAR at `META-INF/openejb-jar.xml`
- EJB settings for CMP/CMR, JNDI/CORBA/Web Service settings for remote clients, MDB configuration
- Otherwise pretty standard (dependencies, resource/EJB/Web Service references, security, gbeans, etc.)

EJB CMP Settings

- DB syntax mapping & DDL generation
- Table/column mappings
- Resolving unknown primary keys
- Automatic PK generation
- Prefetch groups
- Query tuning

EJB CMR Settings

- Maps one-to-one and one-to-many relationships using foreign keys
- Maps many-to-many relationships using a join table
- Can set prefetch group to use when a CMR field is accessed, including multiple levels at once

META-INF/openejb-jar.xml

```
<openejb-jar
  xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.0"
  configId="MyEJBJarName">

  <dependency ...>
  <!-- some CMP settings here -->
  <enterprise-beans>
    <session ...>
    <entity ...>
    <message-driven ...>
  </enterprise-beans>
  <relationships ...>
  <security ...>
  <gbean ...>
</openejb-jar>
```

J2EE Connectors

- Plan in RAR at `META-INF/geronimo-ra.xml`
- Configures instances of the resource adapter, connection factory instances, and admin objects
  - Database: connections to multiple DBs, with same or different drivers
  - JMS: connection factories & destinations

Inbound Connectors

- Configure the thread pool (WorkManager) and connectivity to the messaging server
- Configure destinations that can be accessed individually or mapped to MDBs
- Supports any connector, JMS or otherwise
- Ships with ActiveMQ resource adapter for JMS connections and destinations

Outbound Connectors

- Support connection pools (single pool, subpools per user, etc.)
  - Configurable timeout for a caller to wait for a connection
  - Configurable timeout to reclaim connections in the pool
- Ships with TranQL adapter for JDBC pools

Connector Strategies

- Normally deployed as a top-level module (a server-wide JDBC pool, etc.)
  - This is how the console does it
- Can also package it within an EAR, so the DB pool or JMS resources are deployed and undeployed with the application
  - Still visible to other applications though

META-INF/geronimo-ra.xml

```
<connector
  xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.0"
  configId="MyConnectorName">

  <dependency ...>
  <resourceadapter>
    <resourceadapter-instance ...>
    <outbound-resourceadapter>
      <connection-definition>
        <connectiondefinition-instance ...>
      </connection-definition>
    </outbound-resourceadapter>
  </resourceadapter>
  <adminobject ...>
  <gbean ...>
</connector>
```

J2EE Application EARs

- Plan in EAR at `META-INF/geronimo-application.xml`
- Can point to a module's Geronimo deployment plan inside the EAR but outside the module JAR, or can just put the whole module deployment plan in here
- Can specify dependencies and security settings for all the modules in one shot

Sample EAR Contents

```
my-app.ear/
   my-web.war
   my-ejbs.jar
   tranql-connector-1.1.rar
   some-3rd-party-library.jar
   plans/
      web.xml
      ejb-jar.xml
      geronimo-web.xml
      geronimo-ejb-jar.xml
      dbpool-definition.xml
```

.../geronimo-application.xml

```
<application
xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.0"
  configId="MyApplicationName">

  <dependency ...>
  <module>
    <connector>tranql-connector-1.1.rar</connector>
    <alt-dd>plans/dbpool-definition.xml</alt-dd>
  </module>
  ...
  <security ...>
  <gbean ...>
</application>
```

Summary

Closing Thoughts

- A complete J2EE server
  - Apache Licensed
  - Extremely customizable
- Configuration and DB/JMS/Security setup through the web console
- Deployment tool, Maven deployment plugin, and hot deploy directory
- Deployment plans for J2EE modules
- Can pack resources & services in an EAR

Tomorrow's Workshop

- Hands-on exercises
  - Bring your laptop w/Java IDE, etc.
- Will work through:
  - Installation & basic configuration
  - Using the admin console & deploy tools
  - Deploying a database connection pool, JMS resources, security realm
  - Configuring a web application, EJBs, EAR with several passes and different options

Q&A

E-Mail Lists

- user@geronimo.apache.org
  - subscribe-user@geronimo.apache.org
- dev@geronimo.apache.org
  - subscribe-dev@geronimo.apache.org

IRC

- #geronimo on irc.freenode.net